Tijdschrift voor Economie en Management
Vol. L, 4, 2005

Event-gebaseerde integratie van bedrijfsprocessen met behulp van web services

door W. LEMAHIEU en M. SNOECK

Wilfried Lemahieu
KULeuven, Departement Toegepaste Economische Wetenschappen, Groep Informatica, Leuven

Monique Snoeck
KULeuven, Departement Toegepaste Economische Wetenschappen, Groep Informatica, Leuven

ABSTRACT

Web services vormen een recente, Internet-gebaseerde technologie voor de integratie van applicaties en bedrijfsprocessen binnen eenzelfde onderneming of overheen meerdere ondernemingen. Dit artikel schetst eerst de basisbegrippen van web services en stelt vervolgens een *event-gebaseerde* benadering voor van de integratieproblematiek. Hierbij worden *bedrijfsgebeurtenissen* (*business events*) beschouwd als "atomaire eenheden van activiteit". Deze event-gebaseerde benadering resulteert in een meer accurate procesmodellering en coördinatie tussen web services van verschillende, collaborerende partners.

Web services are a recent, Internet based technology to integrate applications and business processes within a single company and across multiple companies. This paper first sketches the basic principles that underlie web services and then proposes an *event based* integration approach, where *business events* are considered as "atomic units of activity". The event based approach allows for more accurate process modelling and coordination among web services belonging to independent, collaborating partners.

I. NOOD AAN INTEGRATIE BINNEN EN BUITEN DE ONDERNEMING

Eén van de meest toonaangevende ICT-trends van de afgelopen jaren is ongetwijfeld de nood aan *integratie*. Gedreven door nieuwe vormen van bedrijfsvoering, waarin accurate en tijdige informatie over productieprocessen, productiemiddelen én de klant cruciaal is, ontstond de behoefte aan integratie van applicaties zoals Enterprise Resource Planning (ERP), Supply Chain Management (SCM) en Customer Relationship Management (CRM). Dergelijke integratie van toepassingen *binnen eenzelfde onderneming* wordt omschreven met de verzamelterm *EAI (Enterprise Application Integration)*. Twee soorten middleware bieden ondersteuning voor EAI. In de eerste plaats zijn er de RPC (Remote Procedure Call)-gebaseerde middleware-technologieën, die de interactie tussen twee applicaties voorstellen als een *procedure-aanroep* van de ene applicatie op de andere. In toenemende mate worden object-georiënteerde varianten van dit principe gebruikt. Zij maken het mogelijk dat een object een methode aanroept van een ander (remote) object, dat zich op een andere server bevindt. Elk object kan daartoe een remote interface publiceren, die aangeeft welke methoden van op afstand kunnen aangeroepen worden. RPC-gebaseerde technologieën zijn typisch *synchroon* (de aanroeper wacht met verdere uitvoering tot hij een antwoord gekregen heeft) en vereisen een vrij sterke koppeling tussen de interagerende objecten en applicaties. Een tweede soort technologie is MOM (Message Oriented Middleware). Hierbij wordt de interactie voorgesteld als een *boodschap* die wordt uitgewisseld tussen twee of meer applicaties. In tegenstelling tot RPC is MOM-gebaseerde interactie doorgaans *asynchroon*: de aanroeper wacht niet op een eventueel antwoord voor hij verdergaat, zodat systemen niet noodzakelijk tegelijkertijd beschikbaar moeten zijn of aan hetzelfde "tempo" hoeven te werken, wat een veel lossere koppeling toelaat.

Waar EAI-technologieën zich beperken tot integratie binnen eenzelfde onderneming, neemt hand over hand ook de noodzaak toe aan integratie-technieken *over verschillende ondernemingen heen*. Een geoptimaliseerd beheer van bedrijfsprocessen overheen de gehele waardenketen, efficiënte en geautomatiseerde interactie met klanten en toeleveranciers, uitbesteding van specifieke onderdelen van het productieproces aan gespecialiseerde partners, … het zijn allemaal evoluties die vereisen dat verschillende onafhankelijke ondernemingen samenwerken als een "extended enterprise", waarbij hun informatiesystemen tot op zekere hoogte geïntegreerd worden. Het Internet is uiteraard het medium bij uitstek om dergelijke *B2Bi (Business-to-Business integration)* te bewerkstelligen. Helaas bleken zowel de voorgenoemde RPC- als MOM-technologieën te "zwaar" om buiten het locale netwerk van eenzelfde bedrijf te gebruiken én waren ze vrijwel niet compatibel met de huidige generatie firewalls. De enige oplossing was dus een nieuw protocol te ontwikkelen dat "licht" genoeg is om overheen de bestaande Internet-protocols te gebruiken en dat geen firewall-problemen oplevert. Het SOAP-protocol (Simple Object Access Protocol)

komt aan deze eisen tegemoet. De diensten die men via het SOAP-protocol overheen het Internet kan aanspreken noemt men “web services”.

De basisbegrippen voor B2B integratie van bedrijfsprocessen op basis van web services worden uiteengezet in Sectie II. Sectie III stelt in de plaats van de huidige, binaire interactie tussen web services een event-gebaseerd, m-op-n interactiemechanisme voor. Er wordt aangetoond hoe dit mechanisme het niet enkel mogelijk maakt om de geïntegreerde processen nauwkeuriger en eenvoudiger voor te stellen, maar hoe door contractanalyse tevens kan worden nagegaan of de processen die in de respectievelijke services worden belichaamd wel degelijk verenigbaar zijn en welk globaal gedrag hieruit zal resulteren. Sectie IV formuleert conclusies en geeft indicaties voor toekomstig onderzoek.

II. INTEGRATIE VAN BEDRIJFSPROCESSEN MET BEHULP VAN WEB SERVICES

A. *Basisbegrippen van web services*

Een van de meest populaire definities van web services is die van Wahli (2002): “*Web services are self contained, self-describing, modular applications that can be published, located, and invoked across the Web”*. Bij web services in hun meest eenvoudige vorm zijn drie specificaties van belang: SOAP als interactieprotocol, WSDL voor de definitie van web service interfaces en UDDI voor het adverteren van en zoeken naar web services. SOAP, WSDL en UDDI worden hierna kort besproken.

SOAP (zie Seely en Sharkey (2001)) stelt web services in staat om met elkaar te interageren door XML-berichten uit te wisselen. XML (eXtensible Markup Language) is een taal om gestructureerde documenten of berichten aan te maken (zie Yergeau et al. (2004)). In SOAP stelt dergelijk XML-bericht een *methode-aanroep* en de bijbehorende *parameters* voor. Het stelt de ene web service in staat een methode aan te roepen op een andere web service en kan zo beschouwd worden als een lichtere variant van het RPC-protocol. SOAP heeft echter als grote voordeel tegenover respectievelijk traditionele MOM- en RPC-middleware dat het werkt op basis van XML- (en dus in feite tekst-) berichten, die verstuurd worden overheen bestaande Internetprotocols zoals SMTP of HTTP. Daardoor kent SOAP hoegenaamd geen firewall-problemen en hoeft het zich niet te beperken tot locale netwerken, zoals het geval is met voornoemde technologieën. Op deze wijze hebben web services en SOAP op dit ogenblik vrijwel het monopolie op gebied van B2Bi overheen het Internet.

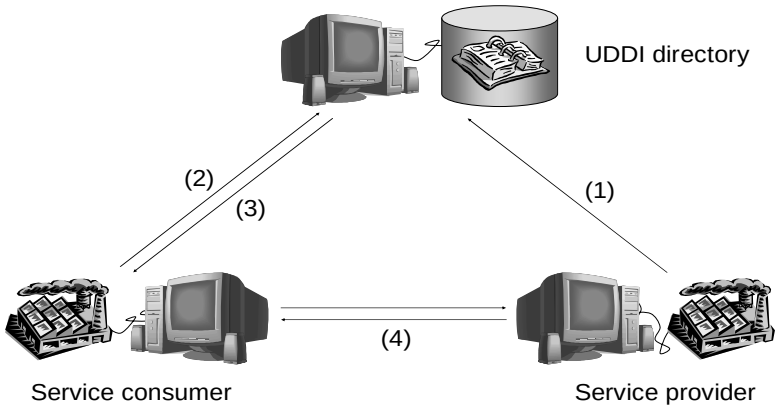
Om zinvolle SOAP-berichten naar een web service te kunnen sturen, moet deze uiteraard eerst meedelen welke soorten berichten hij “begrijpt”. Dat is de taak van WSDL (Web Services Description Language) (zie WSDL (2001)). WSDL is een XML-gebaseerde taal om de *interface* van web

services te beschrijven. WSDL beschrijft voor elke service een aantal abstracte *operaties* die er op aangeroepen kunnen worden. Elke operatie bestaat uit een *input*-bericht en (optioneel) een *output*-bericht met eventuele bijbehorende foutenboodschappen. Van elke soort bericht worden de *attributen* en de overeenkomstige *datatypes* vastgelegd. Deze benadering is vrij gelijkaardig aan het concept "remote interface" dat we reeds kennen van RPC-middleware: de WSDL-beschrijving geeft aan welke operaties op de web service kunnen aangeroepen worden via het Internet en (optioneel) welk soort antwoord kan verwacht worden.

Een Web service kan uiteraard maar succesvol zijn wanneer potentiële gebruikers hem kunnen *vinden*. Dat is de taak van UDDI (Universal Description, Discovery and Integration) (zie UDDI (2000)). UDDI vormt een *directory* van web services, zeg maar een soort "gouden gids" waarin service providers de door hen aangeboden web services kunnen registreren en er een aantal kenmerken van vastleggen. Deze kenmerken omvatten de coördinaten van de onderneming die de service aanbiedt zoals naam, telefoonnummer, adres, enz. en een aantal classificaties, die de service indelen in klassen op basis van industrietak, geografische locatie,… Een laatste belangrijk kenmerk dat wordt opgeslagen is de specificatie van de *interface* van de web service, vastgelegd in een WSDL-document. Vervolgens kunnen service consumers zoeken naar de door hen benodigde service(s) op basis van de hierboven genoemde kenmerken. Algemeen verloopt de interactie met behulp van web services als volgt, zoals geïllustreerd in Figuur 1:

- een service provider registreert de beschrijving van zijn service in de UDDI directory (1)
- een service consumer zoekt naar de gewenste service in de UDDI directory (2) en haalt de WSDL-beschrijving van de gewenste service op (3)
- de service consumer interageert vervolgens met de gekozen web service door de in diens WSDL-beschrijving gespecificeerde operaties aan te roepen (4)

FIGUUR 1
Ontdekken en aanroepen van web services



B. *Definitie van bedrijfsprocessen als choreografieën van web services*

De web services-technologieën die we tot nu toe besproken hebben, maken het mogelijk om een methode-aanroep uit te voeren tussen twee applicaties overheen het Internet. Op deze wijze kan bijvoorbeeld een applicatie van een potentiële klant de prijs van een bepaald product opvragen bij de web service van een leverancier. Om tot echte geïntegreerde e-business-toepassingen te komen, waarbij verschillende partners deelnemen in complexe transacties, is er echter veel meer nodig dan dergelijke eenvoudige request/response communicatie. Een e-business transactie kan niet uitsluitend beschreven worden op basis van individuele aanroepen tussen web services; ze zal doorgaans een langdurig *proces* beslaan. Een aankoopproces zal bijvoorbeeld bestaan uit achtereenvolgens het creëren van een bestelorder, verzending van de bestelde goederen, facturatie, betaling enz. Een bedrijfsproces wordt zo voorgesteld als een verzameling op elkaar volgende (of soms ook parallelle) *activiteiten*. Elke individuele activiteit kan worden gerealiseerd door een bepaalde operatie-aanroep op een web service. Het is niet enkel van belang dat elke individuele aanroep overeenkomt met de WSDL-beschrijving van de betrokken web service, het geheel van de aanroepen dient ook nog in de juiste *volgorde* te gebeuren, zoals bepaald door de bedrijfsprocessen. Zo zal het bedrijfsproces van de ene leverancier stellen dat levering automatisch volgt na de bestelling, waarna de facturatie geschiedt, terwijl een andere leverancier kan opleggen dat de betaling aan de levering dient vooraf te gaan.

Een onderneming zal dan ook niet enkel de WSDL-beschrijving van de interfaces van haar web services moeten publiceren, maar zal daarnaast ook moeten aangeven in welke volgorde de aanroepen naar de respectievelijke interfaces moeten gebeuren om tot een interactie te komen die “past” binnen de bedrijfsprocessen van de onderneming. Dit noemt men een *choreografie* van web services: ze bepaalt de volgorde waarin activiteiten van een bedrijfsproces - en dus de overeenkomstige individuele web service

aanroepen - moeten gebeuren om de bedrijfsprocessen op correcte wijze uit te voeren. Ook deze choreografie-beschrijving wordt in een XML-document vastgelegd. Er zijn op dit ogenblik twee grote kandidaten om tot standaard uit te groeien: BPML (zie Arkin (2002)) en BPEL4WS (zie Andrews (2003)). Beiden hebben vrij gelijkaardige eigenschappen; een globaal proces overheen de extended enterprise wordt beschreven als een web service choreografie, waarbij elke activiteit is voorgesteld als een individuele aanroep van een operatie op de web service van een bepaalde partner. Zo zal bijvoorbeeld een operatie-aanroep van klant naar leverancier een bestelorder voorstellen, een daaropvolgende aanroep van leverancier naar klant zal een bevestiging van verzending van de bestelde goederen vormen, enz. De choreografie-beschrijving legt de volgorde vast waarin die aanroepen dienen te gebeuren.

III. EVENT-GEBASEERDE CHOREOGRAFIE VAN WEB SERVICES

A. *Van één-op-één naar m-op-n interactie*

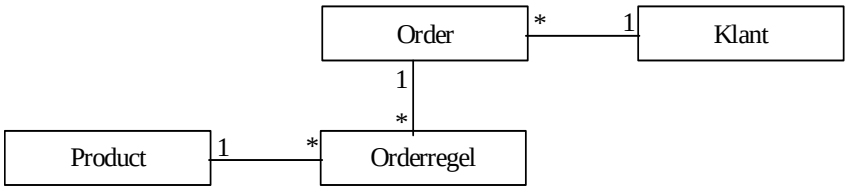
Benaderingen zoals BPEL4WS en BPML zijn in essentie gebaseerd op *workflow* concepten, waarbij de output van de ene service als input van een volgende service wordt gebruikt. Op deze wijze wordt een proces waarin verscheidene partners samenwerken opgesplitst in één-op-één interacties tussen aanroeper en aangeroepene, bijvoorbeeld een klant die een “plaats-bestelling”-operatie aanroept bij een leverancier. Tevens wordt geen onderscheid gemaakt tussen de synchronisatie van *activiteiten* en de *data-uitwisseling* die daarvoor nodig is: beiden worden belichaamd in eenzelfde SOAP-boodschap. In het vervolg van dit artikel stellen wij als alternatief een *event-gebaseerde* benadering voor, om choreografieën van web services te beschrijven. Deze blijven niet beperkt tot één-op-één interactie tussen twee services, maar maken het rechtstreeks mogelijk om een complex proces, waaraan meer dan twee services deelnemen, in een aantal “gebeurtenissen” te beschrijven. Tevens wordt een duidelijk onderscheid gemaakt tussen het synchronisatie- en choreografie-aspect (*event notification*) en het lezen van gegevens (*querying*). In Sectie III.B worden de basisbegrippen van event-gebaseerde interactie besproken. Sectie III.C past deze toe op een web services-omgeving. Sectie III.D geeft aan hoe deze benadering het mogelijk maakt om procesbeschrijvingen te analyseren en de compatibiliteit van web services op dit vlak na te gaan.

B. *Basisbegrippen van event-gebaseerde interactie*

De event-gebaseerde benadering voorgesteld in dit artikel vloeit voort uit de object-georiënteerde analyse- en designmethode MERODE (zie Snoeck

(2003), Snoeck et al. (1999) en Snoeck en Dedene (1998)). MERODE stelt een informatiesysteem voor door middel van *bedrijfsgebeurtenissen* (business events), hun effect op *bedrijfsobjecten* (enterprise objects) en de daaraan gerelateerde *bedrijfsregels*. In tegenstelling tot klassieke object-georiënteerde methodologieën, wordt geen gebruik gemaakt van *methode-aanroepen* om de interactie tussen (klassen van) bedrijfsobjecten te modelleren. In plaats daarvan worden bedrijfsgebeurtenissen als onafhankelijke concepten geïdentificeerd, waarbij een *object-event tabel* aangeeft welke types objecten betrokken zijn bij welke types gebeurtenissen. Laten we bijvoorbeeld veronderstellen dat het domeinmodel voor een orderverwerkingssysteem vier object types bevat: KLANT, ORDER, ORDERREGEL en PRODUCT, zoals voorgesteld in het klassediagramma in Figuur 2. Dit klassediagramma maakt gebruik van de Unified Modeling Language (UML, zie Object Management Group (2003)) om de volgende realiteit voor te stellen: een ORDER bestaat uit verscheidene ORDERREGELS; in elke ORDERREGEL wordt één welbepaald PRODUCT besteld. Een PRODUCT kan in meerdere ORDERREGELS (en dus in meerdere ORDERS) voorkomen. Elk ORDER hoort bij één welbepaalde klant, maar een klant kan meerdere uitstaande ORDERS hebben.

FIGUUR 2
Domeinmodel voor een orderverwerkingssysteem



In het bovenstaande voorbeeld zijn mogelijke gebeurtenistypes *creëren_klant, wijzigen_klant, beëindigen_klant, creëren_order, wijzigen_order, beëindigen_order, annuleren_order, verzending, facturatie, betaling, creëren_orderregel, wijzigen_orderregel, beëindigen_orderegel, creëren_product, wijzigen_product* en *beëindigen_product.* De object-event tabel (zie Figuur 3) toont welke objecttypes betrokken zijn bij welke gebeurtenistypes en geeft ook de soort betrokkenheid aan: C voor creatie, M voor een (mogelijke) wijziging (modification) en E voor het beëindigen (ending) van de levenscyclus van een object. Bijvoorbeeld, *creëren_orderregel* leidt tot de aanmaak van een nieuwe instantie van de klasse ORDERREGEL, wijzigt een instantie van de klasse PRODUCT (omdat het voorraadniveau van het bestelde product dient aangepast), wijzigt de status van het betrokken ORDER én wijzigt de status van de betrokken KLANT. Een meer gedetailleerde beschrijving van de modaliteiten i.v.m. de aanmaak van een object-event tabel en de validatie ervan tegenover datamodel en

gedragsmodel (cfr. infra) vallen buiten het bestek van dit artikel, maar zijn te vinden in (Snoeck et al. (1999)).

FIGUUR 3
Object-event tabel voor het orderverwerkingssysteem

	KLANT	ORDER	ORDERREGEL	PRODUCT
creëren_klant	C			
wijzigen_klant	M			
beëindigen_klant	E			
creëren_order	M	C		
wijzigen_order	M	M		
beëindigen_order	M	E		
annuleren_order	M	E		
verzending	M	M		
facturatie	M	M		
betaling	M	M		
creëren_orderregel	M	M	C	M
wijzigen_orderregel	M	M	M	M
beëindigen_orderregel	M	M	E	M
creëren_product				C
wijzigen_product				M
beëindigen_product				E

Het gedragsmodel bepaalt dat elk objecttype een operatie dient te hebben voor elk gebeurtenistype waaraan het kan deelnemen. Dergelijke operatie implementeert de statuswijzigingen (m.a.w. wijzigingen van attribuutwaarden) van het betrokken object tengevolge van het optreden van een overeenkomstige gebeurtenis.

Daarnaast zijn bedrijfsobjecten ook in staat om *beperkingen* (*constraints*) op te leggen op de gebeurtenissen waarin ze deelnemen. Deze constraints worden afgeleid van de *precondities* die een objecttype kan opleggen op een bepaald gebeurtenistype. Bijvoorbeeld, wanneer een klant een product bestelt, wordt een nieuwe ORDERREGEL gecreëerd door een *creëren_orderregel* bedrijfsgebeurtenis uit te lokken. Precondities voor dergelijke gebeurtenis, zoals bepaald door de respectievelijke participerende objecten, zijn bijvoorbeeld:

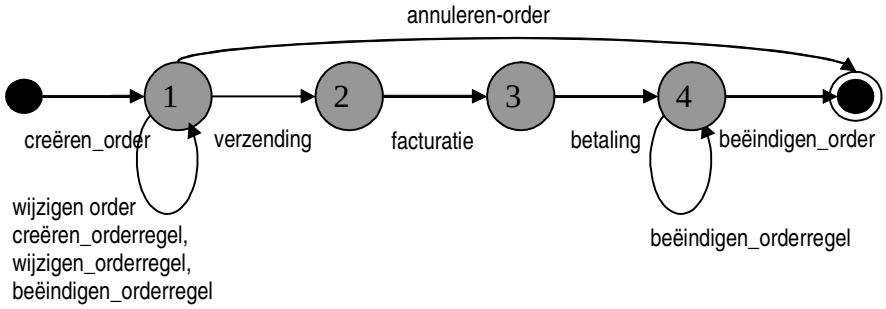
- ORDERREGEL: de betrokken orderregel mag nog niet bestaan;

- KLANT: een klant mag enkel bijkomende producten bestellen als de totale prijs van nog niet betaalde orders een bepaalde limiet niet overschrijdt;
- PRODUCT: een orderregel kan enkel worden aangemaakt als het betrokken product nog voldoende in voorraad is;
- ORDER: een orderregel voor de bestelling van een bepaald product kan enkel worden aangemaakt als het betrokken order nog geen bestelling voor dit product bevat.

Op deze wijze kunnen verscheidene bedrijfsobjecten (vier in het voorbeeld) samen deelnemen aan eenzelfde bedrijfsgebeurtenis en er elk hun eigen beperkingen aan opleggen. Gebeurtenissen implementeren zo op eenvoudige wijze een m-op-n interactie. Dit in tegenstelling tot zuivere methode-aanroepen tussen objecten, die per definitie binair zijn: van aanroeper naar aangeroepene. Bij een binaire interactiepatroon zouden er verscheidene onafhankelijke methode-aanroepen nodig zijn om een KLANT-, PRODUCT- en ORDERREGEL-object te laten deelnemen aan de creatie van een ORDERREGEL. Daarbij is er geen garantie voor de automatische coördinatie tussen deze onderscheiden aanroepen, terwijl ze in principe toch als één coherent geheel moeten beschouwd worden.

Het event-gebaseerde interactieparadigma leent zich verder ook uitstekend om bedrijfsprocessen te modelleren. Een specifieke soort precondities zijn namelijk de *volgordebeperkingen* (*sequence constraints*), die kunnen afgeleid worden uit de zogenaamde *toestandsdiagramma's* (finite state machines) die met elk objecttype geassocieerd zijn. Figuur 4 toont bijvoorbeeld de finite state machine van het ORDER bedrijfsobject. Zolang het niet verzonden werd, blijft een order in toestand 1 (wijzigbaar). In dit stadium is het nog steeds mogelijk het order te wijzigen door orderregels toe te voegen, te wijzigen of te verwijderen. De *verzending* gebeurtenis brengt het order in toestand 2 (geleverd). Vanaf dan kan het order niet meer gewijzigd worden. Dit betekent dat de gebeurtenissen *wijzigen_order*, *creëren_orderregel, wijzigen_orderregel en beëindigen_orderregel* niet langer worden aanvaard voor dit order. De *facturatie* gebeurtenis geeft aan dat een factuur werd verzonden naar de klant. Uiteindelijk geeft de *betaling* gebeurtenis aan dat het order betaald werd. Vanaf dan is het order klaar om verwijderd (of gearchiveerd) te worden. Het *annuleren* van een order is enkel mogelijk in de eerste toestand, wanneer het order reeds bestaat maar nog niet geleverd werd. Op deze wijze weerspiegelen de volgordebeperkingen het achterliggende bedrijfsproces.

FIGUUR 4
State machine voor een order object



Telkens als een bedrijfsgebeurtenis effectief optreedt wordt ze "uitgezonden" (als een soort broadcast- principe) naar alle betrokken bedrijfsobjecten. Enkel gebeurtenissen die voldoen aan de volgordebeperkingen en andere precondities die opgelegd worden door elk van de individuele objecten, kunnen worden geaccepteerd en effectief verwerkt. Bijvoorbeeld, stel dat een *creëren_orderregel* gebeurtenis wordt geïnduceerd, waarin de bestelling voor een bepaald product aan een order wordt toegevoegd. Deze gebeurtenis kan enkel doorgang vinden als ze aanvaard wordt door de betrokken ORDERREGEL (de orderregel mag nog niet bestaan), de betrokken ORDER (volgordebeperking en test of het product nog niet besteld was), de betrokken KLANT (test of het limietbedrag voor niet-betaalde orders niet overschreden is) en het betrokken PRODUCT (test of de voorraad voldoende is). Enkel indien alle objecten instemmen met de gebeurtenis wordt ze effectief verwerkt, waarbij de overeenkomstige methoden worden uitgevoerd en de nodige aanpassingen gebeuren in de status van de betrokken objecten.

Het geheel van bedrijfsobjecten en bedrijfsgebeurtenissen die samen de bedrijfsprocessen vormgeven wordt de *bedrijfslaag* (*enterprise layer*) genoemd. Het uiteindelijke informatiesysteem wordt gerealiseerd als een laag boven de bedrijfslaag, bestaande uit output en input services. *Output services* lezen de attributen van enterprise objecten en bieden deze informatie aan aan de gebruiker. Wanneer een (bedrijfs-)gebeurtenis zich voordoet in de reële wereld verzamelen *input services* invoergegevens van de gebruiker en induceren ze een overeenkomstige gebeurtenis in de bedrijfslaag, wat uiteindelijk leidt tot een aanpassing van de status van de deelnemende bedrijfsobjecten.

C. *Event-gebaseerde interactie tussen web services*

Hoewel de MERODE-methodologie initieel werd ontwikkeld met het oog op LAN-gebaseerde, standalone applicaties (bijvoorbeeld Lemahieu, Snoeck en Michiels (2003)), kan een gelijkaardige benadering met succes worden toegepast voor de coördinatie en integratie van gedistribueerde applicaties die zijn gepubliceerd als web services. Ook hier zal de event-gebaseerde

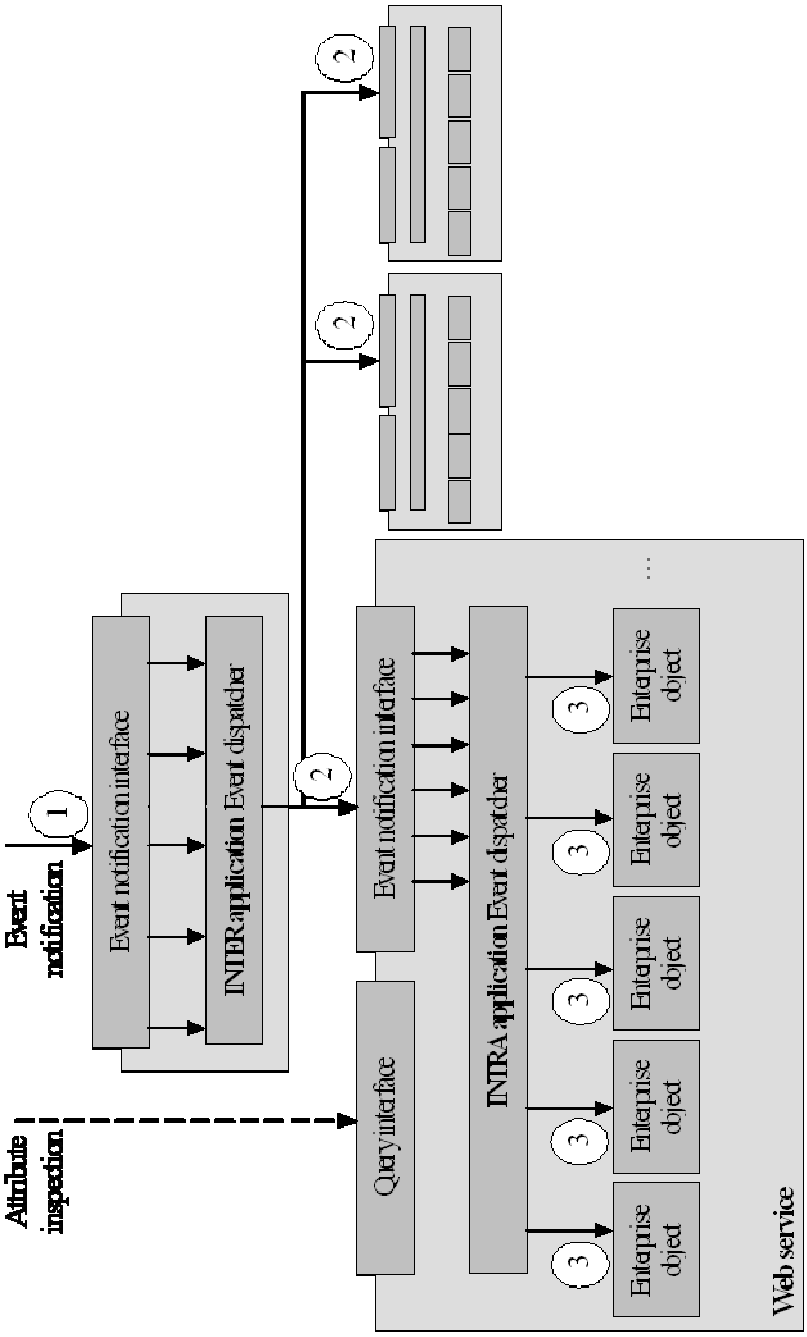
benadering het voordeel opleveren dat gecoördineerde m-op-n interactie tussen web services mogelijk wordt, in plaats van de één-op-één interactie die met de huidige technologie tussen web services plaatsvindt. Tevens wordt het in deze context mogelijk om met behulp van volgordebeperkingen processen te specificeren en dit op een veel efficiëntere wijze dan met flow-gebaseerde voorstellingen zoals BPML en BPEL4WS.

De bedrijfslaag is nu gedistribueerd over meerdere web services heen, waarbij elke web service één of meer achterliggende bedrijfsobjecten verhult. De web service kan daarbij een uniforme, SOAP-gebaseerde interface aanbieden om bedrijfsgebeurtenissen te melden (*de event notification interface*) en om attributen van de onderliggende bedrijfsobjecten te lezen (de *query interface*). Net zoals bij "gewone" web services, worden beide interfaces gespecificeerd met behulp van WSDL. Bedrijfsgebeurtenissen kunnen beschouwd worden als eenheden van "actie" die op één, twee of meer web services betrekking hebben en die in de onderliggende bedrijfsobjecten verwerkt worden.

In een web services-context verloopt dergelijke interactie dan ook op twee niveaus: interactie *tussen* web services gebeurt door de web services te laten participeren aan bedrijfsgebeurtenissen. Met behulp van een *application-event table* kan aangegeven worden welke web service betrokken is in (en moet op de hoogte gebracht worden van) welk type bedrijfsgebeurtenis. Het "broadcasten" van een bedrijfsgebeurtenis gebeurt door een zogenaamde *event dispatcher*. Deze is verantwoordelijk voor de notificatie van alle betrokken web services, coördineert hun antwoorden (succes of mislukking) en bezorgt de component waar de gebeurtenis werd geïnitieerd feedback over de status van de uitvoering. Elke web service die door de event dispatcher op de hoogte is gebracht van een bepaalde gebeurtenis, verwerkt deze gebeurtenis vervolgens intern volgens de eigen bedrijfslogica. Hierbij vindt communicatie plaats *binnenin* de web service, tussen diens interne bedrijfsobjecten. Als een web service intern georganiseerd is om ook volgens een event-gebaseerd interactiemechanisme te werken (wat niet noodzakelijk is), kan de service de gebeurtenis verder intern "dispatchen" naar zijn interne bedrijfsobjecten met behulp van een locale object-event tabel. Dit mechanisme wordt geïllustreerd in Figuur 5: een gebeurtenis wordt geïnduceerd door een aanroep naar de *event notification interface* van een (inter application) event dispatcher (1). Deze *broadcast* de gebeurtenis naar alle web services die bij de gebeurtenis betrokken zijn, door elk van hen aan te spreken via hun event notification interface (2). Elke web service staat in voor de lokale verwerking in zijn eigen bedrijfsobjecten. Indien deze interne interactie eveneens event-gebaseerd is, brengt een intra application event dispatcher de lokale bedrijfsobjecten op de hoogte (3). Deze benadering en de bijbehorende architectuur wordt in meer detail beschreven in (Lemahieu et. al (2003)).

FIGUUR 5

Algemene architectuur voor event-gebaseerde web service interactie



De coördinatie tussen de respectievelijke web services wordt verwezenlijkt doordat elke web service precondities kan opleggen aan de gebeurtenissen waarin hij betrokken is. Bijgevolg kan een gebeurtenis beschouwd worden als een soort *contract* tussen de betrokken web services: elke service legt zijn eigen clausules op door precondities te specificeren. De precondities bepalen de voorwaarden waaraan moet voldaan zijn (vanuit het perspectief van een bepaalde web service) opdat de gebeurtenis doorgang zou mogen vinden. Voor web services waarvan de interne bedrijfsobjecten ook volgens een event-gebaseerd mechanisme interageren, zullen de precondities op web service-niveau afgeleid zijn van de gecombineerde precondities in de onderliggende bedrijfsobjecten die aan de gebeurtenis deelnemen. Het is de taak van de event dispatcher om de atomiciteit van de onderliggende gedistribueerde transactie te garanderen; slechts indien in geen enkele van de participerende web services een preconditie wordt geschaad, zal de gebeurtenis uiteindelijk doorgang vinden en aanleiding geven tot statuswijzigingen in de participerende bedrijfsobjecten. Zoniet wordt de verwerking van de gebeurtenis geweigerd. De taak van de event dispatcher reikt overigens nog verder: hij speelt een belangrijke rol bij het afdwingen van een *proces* overheen de onderscheiden services, zoals besproken in Sectie III.D.

Dergelijk interactiemechanisme maakt het zeer eenvoudig om het aantal web services die aan een bedrijfsgebeurtenis deelnemen te wijzigen: het volstaat om de event dispatcher met een inschrijvings- (subscription-) mechanisme uit te rusten. Wanneer een bepaalde web service belang heeft bij een bepaalde bedrijfsgebeurtenis, kan hij zich inschrijven bij de event dispatcher die dergelijk type gebeurtenissen uitzendt. Als een web service zich inschrijft voor een bepaalde soort gebeurtenis, wordt de overeenkomstige cel in de application-event tabel gemarkeerd. Wanneer een web service niet langer op de hoogte moet gebracht worden van een bepaalde gebeurtenis, volstaat het om de inschrijving op deze gebeurtenis ongedaan te maken door de markering in de overeenkomstige cel van de application-event tabel te wissen.

D. *Analyse van procesbeschrijvingen*

Waar BPML en BPEL4WS web service choreografieën definiëren door de volgorde van één-op-één interacties voor te schrijven in een centraal document, zijn bij de event-gebaseerde benadering de volgordebeperkingen gedistribueerd overheen alle participerende services; elke service kan zijn eigen volgordebeperkingen toevoegen aan een contract. Tevens kan een interactie tussen verscheidene services nu via één gebeurtenis worden voorgesteld, in plaats van te worden opgesplitst in individuele binaire interacties.

Door elke individuele rij in de application-event tabel te beschouwen, krijgen we een overzicht van alle web services die betrokken zijn bij een bepaalde gebeurtenis, alsook een overzicht van alle constraints die worden opgelegd door de respectievelijke deelnemers aan de gebeurtenis. Zoals reeds gezegd bestaan een deel van de beperkingen opgelegd aan gebeurtenissen uit volgordebeperkingen. Web services specificeren volgordebeperkingen door rekening te houden met alle gebeurtenistypes die zich in hun kolom van de application-event tabel bevinden. Deze verzameling gebeurtenistypes noemt men het *alfabet* van de service. Elke web service specificeert de door hem toegelaten sequenties van gebeurtenissen. Dit kan gebeuren met behulp van een reguliere uitdrukking of een finite state machine. Finite state machines identificeren expliciet de verschillende mogelijke toestanden (states) van een component en maken gebruik van gebeurtenissen voor het triggeren van de transities van de ene toestand naar de andere (zie Figuur 4). Reguliere uitdrukkingen focussen zich enkel op de toegelaten scenario's. De respectievelijke toestanden van een component bij het ondergaan van een sequentie van gebeurtenissen worden hier slechts impliciet in aanmerking genomen. Vanuit mathematisch oogpunt hebben beide technieken echter een volledig gelijkwaardige uitdrukkingskracht: voor elke FSM is er een equivalente reguliere uitdrukking en omgekeerd. Zo kan de equivalente reguliere uitdrukking voor de FSM uit Figuur 4 als volgt geformuleerd worden:

ORDER = creëren_order.(wijzigen_order + creëren_orderregel + wijzigen_orderregel + beëindigen_orderregel)* .
(verzending.facturatie.betaling.(beëindigen_orderregel)* .beëindigen_order
+ annuleren_order)

Om de contracten tussen web services op basis van volgordebeperkingen te analyseren, wordt het gedrag van elke service op formele wijze gespecificeerd. Hiertoe wordt gebruik gemaakt van een procesalgebra zoals gespecificeerd in (Snoeck (1995)) en (Snoeck en Dedene (1998)). In deze procesalgebra is elke component gedefinieerd als een paar <□,e>, met □ ⊆ A, waarbij A het universum van gebeurtenissen voorstelt en e een reguliere uitdrukking vormt over A, zodat de gebeurtenissen die in e voorkomen elementen zijn van zijn alfabet □. De reguliere uitdrukking die geassocieerd is met een component definieert de verzameling scenario's die door deze component aanvaard worden. Bijvoorbeeld, de volgende sequenties van gebeurtenissen zijn voorbeelden van scenario's die door het ORDER bedrijfsobject aanvaard worden:

1. creëren_order, annuleren_order
2. creëren_order, wijzigen_order, creëren_orderregel, annuleren_order
3. creëren_order, creëren_orderregel, verzending, facturatie, betaling, beëindigen_orderregel, beëindigen_order

4. creëren_order, wijzigen_order, creëren_orderregel, creëren_orderregel, wijzigen_orderregel, verzending, facturatie, betaling, beëindigen_order

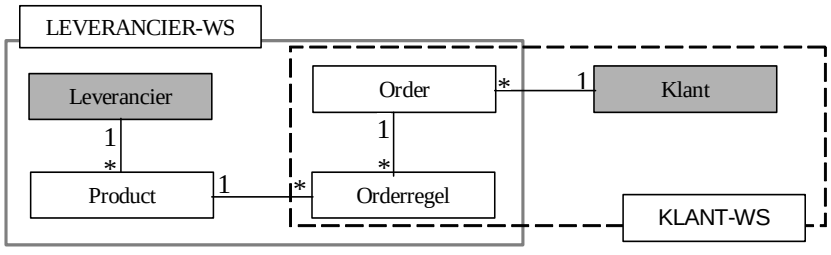
Ditzelfde principe kan ook worden toegepast op het niveau van web services. Veronderstel dat er twee web services bestaan: W = <□,e>, en W' = <□',e'>. De *parallelle compositie* van W en W' vormt een systeem, waarvan het gedrag voorgesteld wordt als W || W'. Wanneer W en W' gezamenlijk deelnemen aan een aantal gebeurtenissen (wat betekent dat □ ∩ □' ≠ ∅), zal het systeem W || W' enkel die scenario's aanvaarden die zowel voor W als W' aanvaardbaar zijn. Formeel betekent dit dat, indien L(W) en L(W') de verzamelingen scenario's voorstellen die door respectievelijk W en W' worden gedefinieerd, dat

$$L(W \parallel W') = \{ s \in (\alpha \cup \alpha')^* \mid s\backslash\alpha \in L(W) \text{ en } s\backslash\alpha' \in L(W') \}$$

waarbij $(\alpha \cup \alpha')^*$ de verzameling van alle mogelijke scenario's voorstelt die gebeurtenissen bevatten van $(\alpha \cup \alpha')$ en $\backslash\alpha$ de projectie-operator voorstelt die alle gebeurtenissen die niet tot α behoren uit het scenario s verwijdert.

In een niet-gedistribueerde omgeving, kunnen de volgordebeperkingen van alle bedrijfsobjecten uit het conceptuele model gezamenlijk worden gespecificeerd, zodat men conflicten bij voorbaat in de modelleerfase kan detecteren. In een gedistribueerde omgeving, zoals het geval is bij web services, worden de volgordebeperkingen onafhankelijk van elkaar gespecificeerd door de verschillende partijen die de respectievelijke web services aanbieden. Het risico op conflicterende volgordebeperkingen is hier dan ook reëel. Laten we volgende voorbeeld beschouwen:

FIGUUR 6
Gedistribueerde architectuur voor een orderverwerkingssysteem

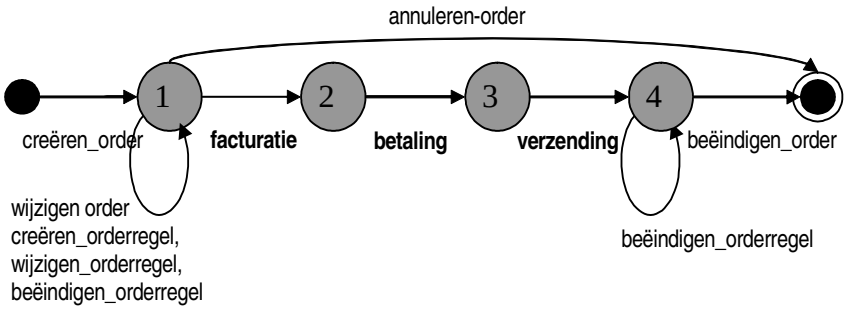


Figuur 6 toont twee web services, die elk een aantal bedrijfsobjecten omvatten. De LEVERANCIER web service biedt de mogelijkheid om producten te bestellen. Deze service houdt informatie bij over alle producten en de

bestelorders die geplaatst zijn voor deze producten. Elke order beschikt over een link naar de web service van de overeenkomstige klant, maar de LEVERANCIER web service houdt zelf geen verdere klanteninformatie bij. De KLANT web service houdt de orders van een bepaalde klant bij. Voor elke order kan de KLANT service een link bijhouden naar de overeenkomstige LEVERANCIER service.

Elke web service beschikt over de mogelijkheid om volgordebeperkingen op te leggen aan de gebeurtenissen waaraan hij kan deelnemen, overeenkomstig het lokale bedrijfsproces van de partner die door de service vertegenwoordigd wordt. Laten we omwille van de eenvoud veronderstellen dat beide web services dezelfde namen gebruiken voor de gebeurtenissen die ze gemeenschappelijk hebben en laten we niet-gemeenschappelijke gebeurtenissen (dus eigen aan slechts één service) buiten beschouwing laten. Indien we bijvoorbeeld veronderstellen dan het bedrijfsproces van de leverancier vereist dat de betaling geschiedt voordat de goederen worden verzonden, dan zien de volgordebeperkingen op order-gebeurtenissen zoals vastgelegd door de leverancier er uit zoals beschreven in Figuur 7. De klant blijft bij de volgordebeperkingen zoals gespecificeerd in Figuur 5. De gebeurtenissen waarin Figuur 7 van Figuur 5 afwijkt, zijn vet gedrukt.

FIGUUR 7
FSM voor een ORDER vanuit het perspectief van de leverancier



Het is duidelijk dat klant en leverancier niet tot een succesvolle transactie zullen kunnen komen. De parallelle compositie van de twee finite state machines toont aan dat de enige scenario's die door de bedrijfsprocessen van zowel leverancier als klant worden geaccepteerd, diegene zijn die eindigen met een annuleren_order gebeurtenis. Dit resultaat kan geformaliseerd worden met behulp van reguliere uitdrukkingen. Het gedrag van de "Klant" web service en de "Leverancier" web service kan als volgt

gedefinieerd worden met betrekking tot hun gemeenschappelijke gebeurtenissen:

```
Klant_ORDER
=  <{creëren_order, wijzigen_order, creëren_orderregel, wijzigen_orderregel,
    beëindigen_orderregel, verzending, facturatie, betaling, beëindigen_orderregel,
    beëindigen_order, annuleren_order},

    creëren_order.(wijzigen_order + creëren_orderregel + wijzigen_orderregel +
    beëindigen_orderregel)* .
    (verzending.facturatie.betaling.(beëindigen_orderregel)*.beëindigen_order
    + annuleren_order)>


Leverancier_ORDER
=  <{creëren_order, wijzigen_order, creëren_orderregel, wijzigen_orderregel,
    beëindigen_orderregel, verzending, facturatie, betaling, beëindigen_orderregel,
    beëindigen_order, annuleren_order},

    creëren_order.(wijzigen_order + creëren_orderregel + wijzigen_orderregel +
    beëindigen_orderregel)* .
    (facturatie.betaling.verzending.(beëindigen_orderregel)*.beëindigen_order
    + annuleren_order)>
```

Berekening van de parallelle compositie van deze twee componenten geeft het volgende resultaat:

```
(Klant_ORDER) || (Leverancier_ORDER)
=  <{creëren_order, wijzigen_order, creëren_orderregel, wijzigen_orderregel,
     beëindigen_orderregel, verzending, facturatie, betaling, beëindigen_orderregel,
     beëindigen_order, annuleren_order},

     creëren_order.(wijzigen_order + creëren_orderregel + wijzigen_orderregel +
     beëindigen_orderregel)*.annuleren_order>
```

Hoewel de resulterende reguliere uitdrukking niet leeg is (wat wel het geval zou zijn bij een volledige deadlock), bevat de uitdrukking niet alle oorspronkelijke gebeurtenissen: de gebeurtenissen *verzending*, *facturatie* en *betaling* ontbreken. Dit wijst er op dat dergelijke gebeurtenissen steeds zullen geweigerd worden door één van de twee web services wegens conflicterende volgordebeperkingen. De “Klant” service zal een *betaling* gebeurtenis weigeren zolang er nog geen *verzending* en *facturatie* gebeurd zijn, terwijl de “Leverancier” de *verzending* zal weigeren zolang er geen *betaling* gebeurd is. Het is duidelijk dat er in deze situatie geen zinvolle samenwerking mogelijk is tussen beide web services.

De berekening van een parallelle compositie kan door een event dispatcher gebruikt worden om de samenwerking tussen meerdere web services te beheren. Tevens is het mogelijk om na te gaan wat er gebeurt als een bijkomende web service zich inschrijft op één of meer gebeurtenissen. Hierbij zou men kunnen opleggen dat enkel web services kunnen inschrijven

waarvan de volgordebeperkingen niet conflicteren met de processen zoals die door de overige reeds ingeschreven web services gedefinieerd zijn. Tenslotte maakt de parallelle compositie het zeer eenvoudig om op een geautomatiseerde wijze een BPEL4WS- of BPML-document te genereren dat het gedrag van het globale systeem beschrijft.

IV. CONCLUSIES EN ONDERWERPEN VOOR VERDER ONDERZOEK

Dankzij de web services-technologie kunnen applicaties van verschillende bedrijven met elkaar interageren over het Internet. De vereiste koppeling blijft zeer los: als enige bindmiddel worden XML-berichten uitgewisseld, die vooraf afgesproken operatie-aanroepen voorstellen. SOAP is daarbij quasi de enige technologie die kan gebruikt worden voor de integratie van informatiesystemen *overheen* bedrijven. Verder bieden web services enkele uiterst belangrijke troeven om werkelijke integratie op het niveau van *bedrijfsprocessen* te realiseren. Hierbij worden collaboratieve processen gedefinieerd als choreografieën overheen web services, waarbij elke web service-aanroep een activiteit in het collaboratieve proces voorstelt.

De voorgestelde event-gebaseerde benadering, waarbij gebeurtenissen de rol vervullen van *contracten* waaraan alle participerende web services precondities kunnen toevoegen, maakt het mogelijk om dergelijke processen rechtstreeks te beschrijven en te coördineren op basis van m-op-n interacties, in tegenstelling tot bestaande choreografie-talen, die zich baseren op één-op-één communicatie tussen web services. De procesbeschrijving wordt voorgesteld door de *volgordebeperkingen* die de respectievelijke web services aan gebeurtenissen kunnen opleggen. De beschreven procesalgebra maakt het mogelijk om de compatibiliteit van deze beperkingen te testen tussen verschillende web services en om vast te stellen of zinvolle transacties mogelijk zijn.

In de huidige incarnatie is enkel de verificatie van volgordebeperkingen mogelijk, verder onderzoek moet toelaten om hier ook andere soorten precondities bij te betrekken. Een ander onderwerp voor toekomstig onderzoek zijn de zogenaamde "*correctieve acties*". Hierbij zal een gebeurtenis die een preconditie schaadt (bijvoorbeeld omdat een product onvoldoende in voorraad is) niet onmiddellijk worden geweigerd, maar zal de betrokken service autonoom trachten het probleem te herstellen door bijkomende, remediërende, gebeurtenissen te induceren (bijvoorbeeld een bijbestelling plaatsen) om zo de oorspronkelijke gebeurtenis toch te laten doorgaan. Hieraan gerelateerd is de mogelijkheid tot *negotiatie*, waarbij incompatibele services door "onderhandeling" tot een voor alle partijen bevredigende procesdefinitie kunnen komen.

REFERENTIES

Andrews T., 2003,, Specification: Business Process Execution Language for Web Services Version 1.1, www-106.ibm.com/developerworks/library/ws-bpel/.

Arkin A., 2002, Business Process Modeling Language, BPMI draft specification.

Lemahieu W., Snoeck M. and Michiels C., 2003, Integration of Third-Party Applications and Web-Clients by Means of an Enterprise Layer, *Annals of cases on Information Technology* 5, 213-233.

Lemahieu W., Snoeck M., Michiels C., Goethals F., Dedene G. and Vandenbulcke J., 2003, Event Based Web Service Description and Coordination, LNCS, Proceedings of the "Web Services, e-Business, and the Semantic Web" Workshop of the CaiSE'03 Conference, (Klagenfurt, Austria).

Object Management Group, 2003, Unified Modeling Language Specification, http://www.omg.org/docs/formal/03-03-01.pdf

Seely S. and Sharkey K., 2001, SOAP: Cross Platform Web Services Development Using XML, (Prentice Hall PTR, Upper Saddle River, NJ).

Snoeck M., 1995, On a Process Algebra Approach for the Construction and Analysis of MERODE-Based Conceptual Models, PhD Dissertation, K.U.Leuven, (Faculty of Sciences and Faculty of Applied Economic Sciences, Department of Computer Science).

Snoeck M., 2003, Bedrijfsmodellering: de sleutel tot de kwaliteit van informatiesystemen, *Tijdschrift voor Economie en Management* 48, 69-96.

Snoeck M. and Dedene G., 1998, Existence Dependency: the Key to Semantic Integrity between Structural and Behavioral Aspects of Object Types, *IEEE Transactions on Software Engineering* 24, 4, 233 - 251.

Snoeck M. , Dedene G., Verhelst M. and Depuydt A., 1999, Object-Oriented Enterprise Modelling with MERODE, (Leuvense Universitaire Pers, Leuven).

UDDI, 2000, Technical White Paper, Ariba, (IBM Corporation and Microsoft Corporation).

Wahli U., 2002, Self-Study Guide: WebSphere Studio Application Developer and Web Services, IBM Redbook, www.redbooks.ibm.com/redbooks/pdfs/sg246407.pdf.

WSDL, 2001, Web Services Description Language 1.0. specification, http://msdn.microsoft.com/xml/general/wsdl.asp.

Yergeau F., Bray T., Paoli J., Sperberg-McQueen C. M. and Maler E., 2004, Extensible Markup Language (XML) 1.0 (Third Edition),W3C Recommendation, http://www.w3.org/TR/2004/REC-xml-20040204/